Legal Intake AI Backend Prototype - Specifications and Design Document

---

1. Overview:

The **Legal Intake AI Backend Prototype** is a comprehensive system designed to automate and streamline a legal services agency's intake and consultation process. This backend prototype enables users to inquire about legal services, book appointments with lawyers, and receive voice-enabled interactions. The system integrates several advanced techniques, including **machine learning**, **natural language processing (NLP)**, **speech-to-text**, and **text-to-speech** to provide an intelligent and responsive user experience.

---

2. Key Features:

- **User Inquiry**: Respond to user inquiries about services, fees, and availability using **FAQ data** or a **dynamic RAG (retrieval-augmented generation)** pipeline for more complex questions.
- **Appointment Booking**: Allows users to book appointments with lawyers, capture client details, and scheduling sessions.
- **Voice Interaction**: Supports voice input and output, enabling users to interact with the system hands-free.
- **Email Confirmation**: Sends appointment confirmations to users through email using a third-party service.

---

3. System Architecture:

The system is composed of several modules, each serving a specific purpose. Below is an architectural breakdown of the components involved:

3.1. Frontend Interaction Layer

- **API Layer**: The FastAPI server acts as the main API endpoint for handling all user interactions.
    - **Endpoints**:
        - /inquiry/: Accepts POST requests with user queries, invokes the get_response function, and returns the response.
        - /voice-inquiry/: Handles voice-based queries by converting speech to text, invoking the AI backend, and speaking the response.
        - /book-appointment/: Accepts POST requests with appointment details, saves the appointment, and sends email confirmation.

3.2. AI Response Generation:

- **FAQ Database (MongoDB)**: Stores predefined FAQs about services, costs, and lawyer availability.
- **RAG Pipeline**: If a question is not found in the FAQ database, the system invokes the RAG (retrieval-augmented generation) pipeline to generate a response using context retrieved from a static legal agency webpage.
    - **Embedding**: Web page content is embedded using **sentence-transformers** and stored in a vector store.
    - **Language Model (LLM)**: Uses **Llama 3.1 (8b)** for processing queries with the relevant context from the retrieved documents.

3.3. Database Layer:

- **MongoDB**: Used for storing **appointments** and **FAQ data**.
    - **Appointments Collection**: Stores client and appointment details.

    - **FAQs Collection**: Stores question-answer pairs to respond to user queries.

3.4. Communication Layer:

- **Email Service**: Sends appointment confirmations via email using **Mailtrap** for testing SMTP configuration.
- **Voice Service**: Utilizes **speech_recognition** and **pyttsx3** to handle voice-based input and output for user interactions.

---

4. Tools and Technologies:

The Legal Intake AI Backend Prototype uses the following technologies:

4.1. FastAPI

- **Purpose**: A modern, fast (high-performance) web framework for building APIs with Python 3.7+ based on standard Python type hints.
- **Role in the System**: FastAPI acts as the main API server, exposing endpoints for user interactions (queries, bookings, voice inquiries).
- **Benefits**: Asynchronous support, easy integration with databases, automatic API documentation generation, high performance.

4.2. MongoDB:

- **Purpose**: A NoSQL database used for storing and retrieving appointment data and FAQ records.
- **Role in the System**: MongoDB stores the appointment details and FAQ questions/answers that can be fetched quickly for user queries.
- **Benefits**: Flexible schema design, scalability, and fast retrieval of documents.

4.3. RAG Pipeline:

- **Purpose**: A technique for generating accurate responses by retrieving relevant information and augmenting it with a language model.

- **Role in the System**: The RAG pipeline retrieves context from a pre-processed static webpage and uses **Llama 3.1** for response generation, ensuring that the AI provides contextually accurate answers for complex inquiries.
- **Technologies**:
    - **WebBaseLoader**: Loads content from static webpages.
    - **HuggingFace Embeddings**: Uses embeddings from sentence-transformers to convert webpage content into vector representations.
    - **SKLearnVectorStore**: A vector store to index and retrieve document embeddings.
    - **Llama 3.1 (8b)**: The language model used to generate responses based on retrieved document context.

4.4. Speech-to-Text and Text-to-Speech:

- **Speech Recognition (speech_recognition)**: Used to convert the user's voice into text.
- **Text-to-Speech (pyttsx3)**: Converts text responses into speech for voice-based interactions.
- **Role in the System**: Provides an interactive, voice-enabled interface for users to ask questions and receive spoken responses.

4.5. Email Service (Mailtrap):

- **Purpose**: A testing tool for simulating email delivery.
- **Role in the System**: Sends appointment confirmation emails to users.
- **Benefits**: No need to worry about actual email sending in the testing phase, easy configuration of SMTP settings.

---

5. Data Flow:

5.1. Voice Interaction:

1. The user speaks into the microphone.
2. The **listen_to_user()** function captures the audio, which is then converted to text using **speech_recognition**.
3. The transcribed query is passed to the **get_response()** function.
4. If the question exists in the MongoDB FAQ database, it returns the corresponding answer.
5. If no match is found, the RAG pipeline is invoked to retrieve relevant information from the pre-processed legal content, and a response is generated by the Llama 3.1 model.
6. The **speak_response()** function converts the text response back into speech and plays it to the user.

5.2. Inquiry via API:

1. A user sends a POST request with a question to the /inquiry/ endpoint.
2. The query is passed to the **get_response()** function.
3. The response is returned from either the FAQ database or the RAG pipeline.

5.3. Appointment Booking:

1. A user sends a POST request with the appointment details to the /book-appointment/ endpoint.
2. The appointment details are saved in MongoDB using the **save_appointment()** function.
3. An email is sent to the user using **send_email()**, confirming the appointment.
4. A success message is returned to the user.

---

8. Conclusion:

The **Legal Intake AI Backend Prototype** is a robust and scalable solution for automating legal services intake and providing seamless interactions between users and the legal services agency. The system combines modern AI, NLP, and speech-processing technologies to offer an intuitive and efficient experience. By leveraging FastAPI, MongoDB, RAG, and other key technologies, this prototype is positioned to provide valuable legal assistance and support to users in a variety of scenarios.

---

9. References:

1. https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2
2. https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct
3. https://www.langchain.com/
4. https://fastapi.tiangolo.com/
5. https://www.mongodb.com/
6. https://mailtrap.io/
